Towards an Infrastructure for MLS Distributed Computing

Myong H. Kang, Judith N. Froscher, and Brian J. Eppinger

Naval Research Laboratory
Information Technology Division
Washington, DC, 20375

Abstract

Distributed computing owes its success to the development of infrastructure, middleware, and standards (e.g., CORBA) by the computing industry. This community has also recognized the need to protect information and has started to develop commercial security infrastructures and standards. The US Government must protect national security information against unauthorized information flow. To support MLS distributed computing, a MLS infrastructure must be built that enables information sharing among users at different classification levels. This infrastructure should provide MLS services for protection of classified information and use both the emerging distributed computing and commercial security infrastructures. The resulting infrastructure will enable users to integrate commercial information technology products into their systems.

In this paper, we examine the philosophy that has led to successful distributed computing among heterogeneous, autonomous components and propose an analogous approach for MLS distributed computing. We identify some services that are required to support MLS distributed computing, argue that these services are needed regardless of the MLS architecture used, present an approach for designing these services, and provide design guidance for a critical building block of the MLS infrastructure.

1. Introduction

The goals of distributed computing have not been achieved without pain. The 80's and early 90's witnessed not only the reengineering of corporate business processes to compete in global markets, but also the migration of those corporations' legacy information technology (IT) assets to widely distributed, interoperable components. This journey has been difficult and many wrong turns were taken. The development of a distributed computing infrastructure, middleware, and standards by industry have resulted in the promise of true interoperability among globally distributed users. These standards have also made it possible for many different vendors to build IT products that are interoperable. IT users, developers, and businesses, generally, have benefited from interoperable products. The US Government policy is to reap these benefits and enjoy similar productivity increases by harvesting COTS products.

In particular, the use of COTS products promises the warfighter global access to open resources. However, the use of COTS products introduces additional risk to national security because COTS products do not address the secure handling of classified information. To provide secure access to both open and classified resources, the DOD must provide multilevel secure (MLS) services to ensure that only properly cleared users access classified national security information. The confidentiality, integrity, and availability of national security resources must be protected both from hacker attacks and from attacks mounted by national intelligence organizations. In a distributed system, lots of data, even code, moves from system to system. How to restrict access to the data and system resources is an important problem. Commercial security services (e.g., CORBA security services [6] and Java security [3]) attempt to address secure information sharing in a single-level distributed system. These security services are designed to work in a heterogeneous environment while preserving other properties such as autonomy and location independence. The DOD should make appropriate use of these services to protect single-level information.

A multilevel secure (MLS) service allows users with different clearances to access all and only the data their clearances authorize them to see. MLS distributed systems must satisfy functional, distribution, and single-level security requirements as well as

- enforce strict separation among classification domains and
- control the flow of information across classification boundaries.

MLS-specific requirements for distributed system design hinge on prevention of unauthorized information flow and require convincing evidence, i.e., assurance, that no high information is released to unauthorized systems and users. High-assurance MLS systems are extremely difficult and expensive to build because the software must satisfy rigorous development standards, must undergo an extensive evaluation and certification process by an independent party, and must be protected against the insertion of malicious code throughout the entire lifecycle. As a result, almost no MLS systems are in operational use today. MLS systems are just not tractable in the current fast-paced development of new technology. Just as distributed computing only became feasible after industry developed infrastructures, middleware, and standards to support it; MLS distributed computing needs an infrastructure, middleware, and standards to make it tractable for operational use. The MLS infrastructure must co-exist with industry standard infrastructures for distributed computing and security, and must provide standard MLS services to support MLS computing.

The role of multilevel security engineers includes devising approaches that can make MLS distributed computing tractable. One way to achieve this goal is

- to develop a distributed system design approach that separates MLS protection from the design of other required functionality,
- to build a MLS distributed computing infrastructure
  - to hide the complexity of the required MLS mechanisms from system designers as much as possible,

  - to separate application specific security requirements from MLS enforcement mechanisms (i.e., MLS infrastructure provides MLS enforcement mechanisms),
  - to allow system designers to use commonly accepted distributed computing services and applications (i.e., The purpose of MLS infrastructure is to extend single-level security services, distributed computing infrastructures and applications across classification domains), and
  - to standardize the approval process for using MLS distributed computing systems,
- to offer cost effective, reusable, and easy to maintain security devices.

In this paper, we propose an infrastructure for MLS distributed computing, and identify some necessary services and critical MLS building blocks for the proposed MLS infrastructure. When we examine the MLS problem, we must be concerned with assurance, which in turn demands clean and simple engineering solutions. Therefore, throughout this paper, we apply software-engineering principles, such as separation of concerns (e.g., trust and functionality) to design a MLS infrastructure and critical MLS building blocks.

1. MLS Distributed Computing

Today's system designers and users have higher expectations than ever for usability and functionality of computer systems. Distributed object computing standards, like CORBA and DCOM, have made a basic level of interoperability possible. For example, in today's distributed object-oriented computing environment, lots of client and server objects reside in many different hosts (i.e., *heterogeneity*). Designers and users of globally distributed objects cannot be expected to know whether server objects are located in the same machine or at a remote host (i.e., *location independence*). Users and organizations want to manage their own data and computing resources (i.e., *autonomy*), but at the same time they expect sharing of information among different organizations across many systems (i.e., *information sharing*). Keeping servers active all the time hogs system resources and can be exploited to deny service to legitimate users. On the other hand, users expect a client object to be able to send a request to a server at any time and receive the reply right away (i.e., *performance and usage of system resources*). To satisfy these high expectation, standards and infrastructures for distributed computing have been built (e.g., CORBA).

What makes MLS protection different from single level security? MLS mechanisms ensure that principals can access all and only the information they are authorized to see. No protection mechanism is perfect. Flaws can be introduced in the lifecycle from the concept stage through implementation and maintenance. Any flaw in a protection mechanism can become a means for illegally leaking information or for inserting false information or malicious code. MLS protection identifies and restricts insecure information flow, for example the flow of more sensitive information to less sensitive users. An assurance argument must be developed to demonstrate that the mechanism is effective, is correctly implemented, and lastly has been so thoroughly evaluated and

analyzed that there is high confidence that we have found all the exploitable vulnerabilities. The development of this assurance evidence and independent evaluation have made MLS products very costly, outmoded, difficult to use and, therefore, not really tractable.

Before we define the services that a MLS distributed computing infrastructure should provide, let us examine some lessons learned from a few successful single-level security-related technologies. Two security technologies have become commonplace in today's distributed computing environment. The first is cryptography, which can provide sender-to-receiver authentication, non-repudiation, and ensure the integrity and privacy of data in transit through a network. The second example is firewall-related technology that can isolate a community of interest from unwanted outsiders. The two main reasons for the success of these security technologies are

1. they satisfy the needs of users reasonably well,
2. their use is independent of system functionality, and therefore, almost transparent to system designers and end users (e.g., a system designer does not have to worry about whether the software will be used inside the firewall or not).

Distributed computing infrastructures have made it much easier for developers of distributed systems to practice the software engineering design discipline: separation of concerns. They must understand and correctly use the infrastructure and its services. When designing a system, the infrastructure allows distributed computing concerns to be addressed separately from functional issues, and makes the development of distributed systems tractable. Similarly, single-level security services must be easy to use and their use must be independent of the desired application functionality.

There are many requirements that MLS system designers have to consider. Some of them are depicted in figure 1. Functional and distribution requirements are not much different from those for single-level distributed systems. Example requirements include supporting heterogeneity, autonomy, location independence, and transparency of the distributed application development process (i.e., developing applications for a distributed system should not be too different from developing applications for a standalone system). Additionally, MLS distributed systems can use some single-level security mechanisms (e.g., authentication, privacy, integrity).

The MLS infrastructure should provide the services and MLS functionality to allow MLS system designers to practice an engineering discipline that is similar to that used for distributed single-level systems. The MLS infrastructure must enable the system functionality to be separate from MLS enforcement and distributed computing concerns. To be successful, MLS enforcement must also be tractable. We believe that being able to reason about security independently makes tractability more attainable.

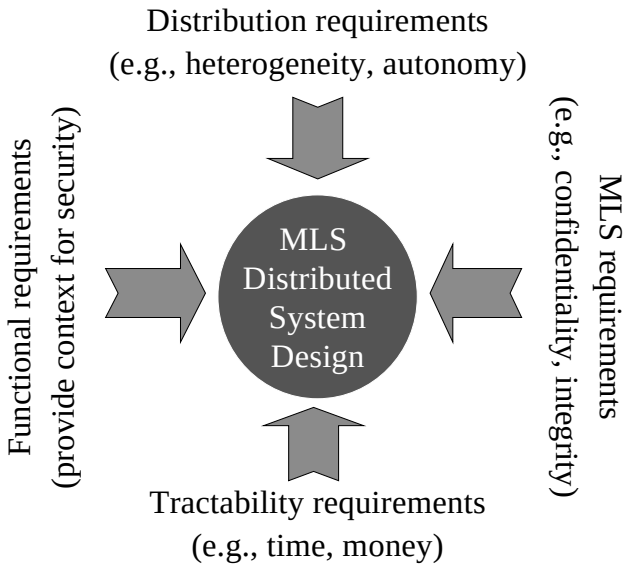

Figure 1: Requirements for MLS distributed system design

In the following subsections, we examine a few high-level requirements for a MLS distributed infrastructure. We also show that these requirements are not specific to a particular MLS architecture.

2.1. A Few Services for a MLS Distributed Computing Infrastructure

MLS distributed computing is still in its infancy. For widespread use of MLS distributed computing, the MLS infrastructure should provide equivalent services and programming paradigms to the single-level distributed computing infrastructure. Learning from the past 20 years of MLS computing history, it is not practical to expect that MLS distributed computing will depend only on a MLS distributed computing infrastructure, which is built from scratch. What we need is a MLS infrastructure that

- seamlessly works with the single-level distributed computing infrastructure,
- operates in a heterogeneous environment while preserving other properties such as autonomy and location independence, and
- provides similar services to those available in a single-level distributed computing infrastructure across classification boundaries.

From these requirements, we derive a few important principles and identify MLS services that the MLS distributed infrastructure should provide. When we derive and design MLS services for the MLS infrastructure, we want to make sure that the MLS system designers can apply widely used design principles and paradigms. Only then will security become an "enabling technology" rather than an "encumbering technology." In this context, we set out the goals in designing the MLS distributed infrastructure.

- The infrastructure should facilitate and encourage system engineers to concentrate on system functionality. This is possible only when the MLS infrastructure faithfully carries out its "behind-the-scenes" support. This is extremely important because

system functionality gives the context in which the security solution has to live and be used.

- The infrastructure should support a sound architecture and consist of well-defined functional units so that the MLS system designer makes the correct choice and can easily show that the MLS distributed architecture is secure. This principle provides a basis for using the appropriate assurance techniques to build different trusted components.
- The infrastructure should support a flexible architecture so that users and designers can place the right functions at the right place. It is important because today's user wants to manage his own computing resources and is responsible for maintaining his own resources (i.e., autonomy).
- The infrastructure should be as transparent as possible in terms of usability, performance, and the consumption of system resources.

Based on the above design goals and recent developments in single-level distributed systems, some necessary services that cross the classification boundaries are identified.

1. *MLS server activation*. There may be many servers that expect requests from clients at different classification domains. Requiring these servers to be active all the time places an extra burden on the systems. Hence, a MLS activation service that can activate the server when requests from other classification domains arrive is needed.
2. *MLS request/reply coordination.* When a client and a server are located in different classification domains, the client's request and server's reply may not go through the same channel. For example, a client's request from a high domain to a lower domain may go through a downgrader, but the server's reply from the lower domain to the high-level client cannot go through a downgrader. However, at the same time, we do not want client software to behave differently when the servers are located in the same classification domain. Hence, there may be a need for a coordinator that can associate the corresponding reply to client's request and direct the reply to the correct client.
3. *MLS cryptography*. A MLS cryptographic infrastructure that can provide authentication and non-repudiation of the senders, and the integrity and privacy of network messages from a sender at one classification domain to a receiver at different classification domain is needed. This infrastructure should provide secure extensions of single-level cryptography across classification boundaries.

2.2. MLS Distributed System Model

In this section, we will show that the MLS services that are described in section 2.1 are not specific to a particular MLS architecture. Rather, they are needed in all MLS distributed systems although the implementation details may vary depending on the MLS distributed architecture. There are three major approaches for building MLS distributed systems. They are multiple single-level systems, distributed MLS-systems, and a hybrid of the two approaches.

2.2.1. Multiple Single Level (MSL) Approach

In this approach, MLS distributed systems are composed of single-level systems and classification boundary controllers that control information flow among systems at different classification levels [2, 4]. In this approach, classification boundary controllers comprise small, high assurance MLS trusted devices with single level policy servers for release and receipt of data. Most other components are mostly untrusted COTS products and a few trusted single-level products. Some single-level products may be trusted to do other tasks like single-level separation and cryptographic certificate and key management. These components are part of the single-level security infrastructure. In this approach, the separation among different classification levels does not have to be physical separation. Data with different classifications can be logically separated through cryptographic means. Figure 2 shows a MSL approach where some systems can communicate with classification boundary controllers directly while others cannot. Whether systems are allowed to communicate with boundary controllers is governed by the release and receipt policies of each community of interest.

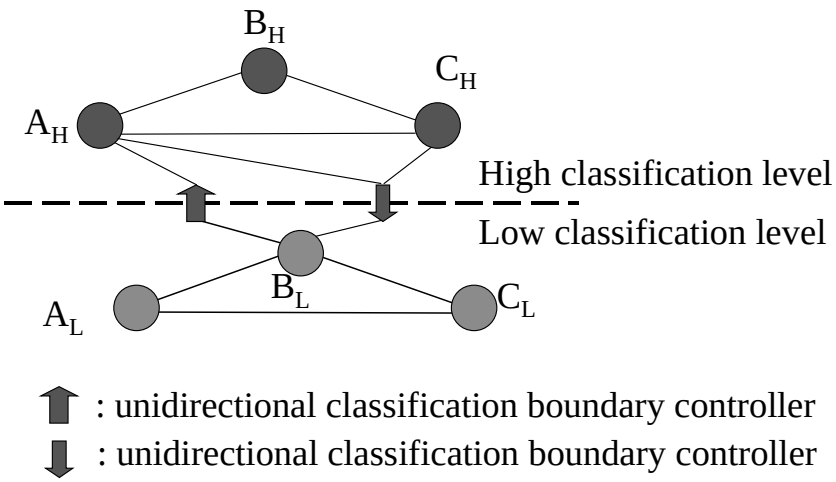

Figure 2: MSL approach to construct MLS distributed systems

Consider a scenario where a client object that resides in host A_H accesses a server object in host B_L through a classification boundary controller. In this simple call, sender-to-receiver authentication, authorization, the integrity and privacy of message in the network are needed. Those properties have to be maintained across classification boundary controllers. Hence, MLS cryptographic services are needed. The server at the low classification level may not be active when the request arrives; hence, server activation services across a classification boundary are needed. If the reply from the server has to go back to the client through an upward information flow controller, then MLS request/reply coordination services are needed to send the right reply to the client that made the request.

2.2.2. Distributed MLS-systems Approach

This approach constructs MLS distributed systems by composing MLS standalone systems. The high portion of a MLS system communicates to the corresponding high portions of other MLS systems. The low portion of a MLS system communicates to the low portion of other MLS systems. The MLS operating system maintains separation among data with different classifications. There is no separate boundary control device in this approach. Nevertheless, each MLS system has a built-in read-down mechanism that allows high-level processes to access lower-level information in the same MLS system. If there is a need to release information (a request is a form of information), the information has to go through a classification boundary controller that is trusted and controls the flow of illegal information in the MLS system.

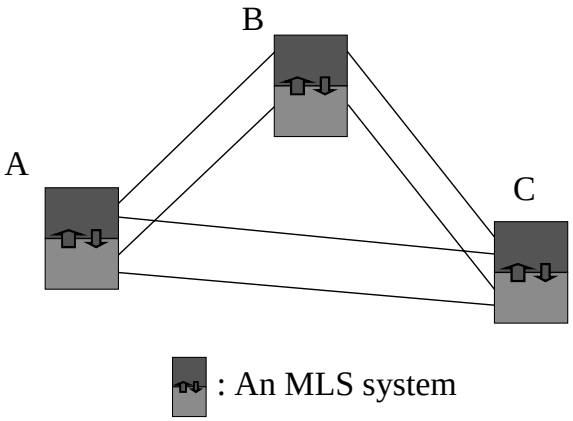

Figure 3: Distributed MLS standalone systems approach

Let us consider the same scenario that we introduced in section 2.2.1 (i.e., a high client object that resides in the high portion of host A accessing a low server object that resides in low portion of host B). It is easy to see that we need the same three services in this MLS distributed system. Note that even though the read-down mechanism allows high clients to read information in a lower-level file or database, it cannot activate lower-level objects to perform a task. In this case, system A or System B can become a classification boundary controller by extending the MLS operating system with trusted code that allows the information flow across classification levels to bypass security policy enforcement.

2.2.3. Hybrid Approach

In this approach, MLS distributed systems are composed of single-level systems and MLS systems that act as classification boundary controllers, MLS servers, or MLS clients. It is easy to see that this approach is very similar to the MSL approach except that the unidirectional classification boundary controllers may be collapsed into a single MLS system. In this approach, a system designer may choose to deploy separate boundary controllers due to assurance, maintainability, availability, performance, or cost reasons.

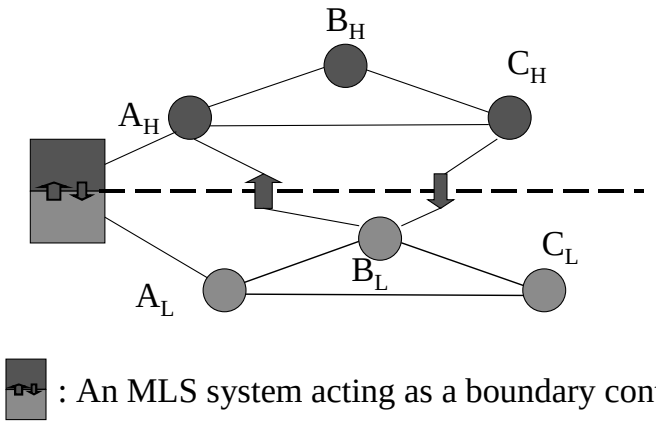

Figure 4: A hybrid approach to construct MLS distributed systems

Let us consider the same scenario that we introduced in section 2.2.1 (i.e., a client object that resides in host A_H accessing a server object in host B_L). Again, it is trivial to see that we need the same three services in this MLS distributed system.

2.2.4. Recap

By examining a simple scenario in the three general approaches to constructing MLS distributed systems, it is easy to arrive at the conclusion*: MLS distributed systems require the same set of services across classification boundaries*. In fact, these services are simply MLS extensions of services needed for single-level distributed computing. So far, the MLS infrastructure should provide classification boundary controllers and support for extending single-level distributed computing and security services across classification boundaries.

Note that some boundary controllers are better suited to the client-server distributed computing paradigm than others. For example, consider the NRL Pump [5] and the read-down capability provided by MLS systems. From a principal's perspective, both can be considered an upward flow controller. The NRL Pump is designed to work in distributed environments in the sense that it supports multiple connections and it "listens" for messages to arrive. However, the read-down mechanism in a MLS system is designed for a stand-alone system where a high-level process "reads down" to access lower level information. If we use the read-down mechanism in a client-server distributed computing paradigm, a high-level process has to poll low-level processes to determine whether messages have arrived at the lower level.

3. Proposed Infrastructure for MLS Distributed Computing

In this section, we propose an infrastructure for MLS distributed computing. We then analyze classification boundary controllers that are the cornerstones of the MLS

infrastructure. In that process, we identify a generic flow controller and present a potential logical design of the flow controller.

Throughout this section, we consistently apply the same guiding principle, separation of concerns (e.g., trust and functionality). We strongly believe that applying this principle to design MLS infrastructure is very important because separation of concerns enables the system designer to produce tractable solutions.

3.1. MLS Service Solutions

In this subsection, we closely look at each of the services for a MLS distributed infrastructure, as defined in section 2.1, and propose solutions. We believe the solutions should meet the following requirements.

- Proposed solutions should be independent of the MLS architecture.
- Proposed solutions should work with security unaware software, COTS or no COTS.
- Proposed solutions should have efficient system resource utilization, pay as little performance penalty as possible, and be easy to manage.

In this subsection, we use the MSL approach as our target architecture. However, it is not difficult to modify the proposed solutions to work with other architectures.

3.1.1. MLS Server Activation Service

In today's distributed computing environment, a lot of server objects are required in order to provide the appropriate system functionality and flexibility. It is not practical for the all these resource servers to always be active and thereby wasting system resources. What we need is a server activation scheme across classification boundaries, which will augment the current single-level services. Figure 5 shows our proposed solution.

Our proposed solution involves the use of MLS activation daemons, which are designed to listen to boundary controllers for service requests. Servers (i.e., proxy servers in our case) that need to be started must first be registered with the MLS activation daemon. After a server has been registered, a (proxy) client can use that server by passing the desired server name to the MLS activation daemon. When the MLS activation daemon receives a request, the daemon will activate the target server and redirect the message traffic to that server.

The approach that we have described thus far is similar to single-level activation services such as Orbix's Orbixd daemon or Java's remote object activation daemon. Single-level clients and servers can talk to each other either through Internet Inter-ORB Protocol (IIOP) or Java's Remote Method Invocation (RMI) protocol. The difference in the MLS case is that those protocols have to pass through a classification boundary controller. One potential solution is to make the classification boundary controllers aware of the IIOP and RMI protocols. The drawbacks of this approach are:

- Whenever a new protocol appears, the classification boundary controllers have to be expanded and may need to be re-evaluated, re-certified, or re-accredited.
- Boundary controllers, in general, do not provide a good programming environment.
- Some protocols may require feedback from the server that can not be accommodated through a boundary controller.

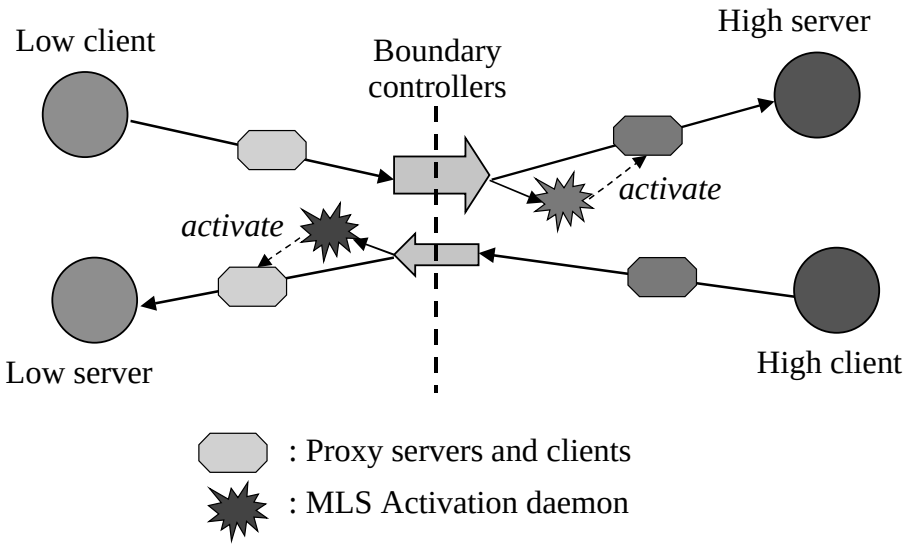

Figure 5: MLS server activation service

There is another way to provide the same capability. Rather than having boundary controllers be aware of all the protocols that potentially pass through them, we make boundary controllers communicate to other software through their own protocol. The translation of application specific protocols to a boundary controller protocol is handled by *proxies*, which we sometimes call *wrappers* that wrap boundary controllers from applications. For example, when a high CORBA client requests some service from a low CORBA server, the high client activates the high proxy through a single-level activation service such as CORBA activation daemon. The high proxy translates the IIOP request to the information release boundary controller protocol. The information release boundary controller then activates the low proxy through the MLS activation daemon to deliver the request. The low proxy translates the request that is in the form of the information release boundary controller's protocol into an IIOP request. It then can activate the real server through Orbixd. We will see how a reply from the server, if needed, can be returned to the client in the next section. Note that we could potentially have one proxy per application protocol.

3.1.2. MLS Request/Reply Coordination Service

Usually in single-level distributed systems, the request and reply paths are through the same logical connection (e.g., a connection-oriented protocol). However, requests from clients and replies from servers may not pass through the same logical connection in MLS distributed systems. To provide an illusion to the client and server objects that they have a bi-directional connection, MLS coordination services are needed. The object or process

that provides such a service is called a "coordinator" in this paper. The main responsibilities of the coordinator include:

- Act as a fake server/client so that the real client/server can establish a connection through a coordinator within the same classification domain to send requests and receive replies.
- Make use of proper classification boundary controllers.
- Coordinate replies for the proper requests.

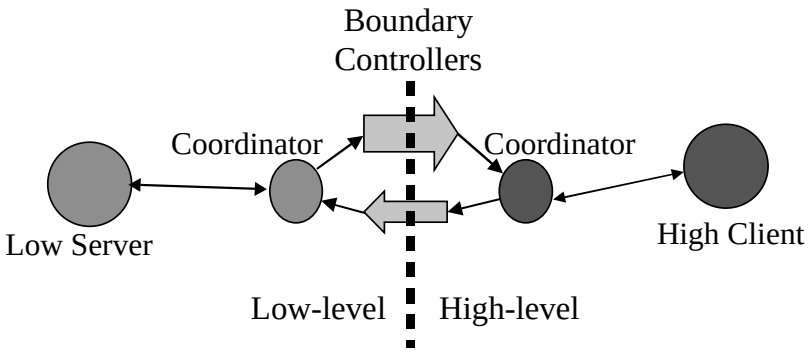

Figure 6: MLS coordination service

Figure 6 provides an architecture for MLS coordination services. If a high client wanted a connection-oriented communication channel with a low side server, then the coordinators would be required to act as proxies to provide that connection. The coordinators would also be required to route traffic from various clients and servers to the correct destinations.

3.1.3. MLS Cryptography

In today's distributed systems, users are interested in the full range of cryptographic services. These services include privacy, authentication, integrity, and non-repudiation. MLS distributed systems, where senders and receivers may reside in different classification domains, require the same security properties. A MLS infrastructure requires a comprehensive solution to provide the equivalent security service across the classification boundaries due to:

1. each classification domain probably has a different sets of principals and
2. each classification domain may use different cryptographic infrastructures (e.g., one classification level uses a Kerberos-based cryptographic infrastructure and another classification level uses SSL-based cryptographic infrastructure).

The above two factors are not unique to MLS distributed computing. However, solutions to those problems are more difficult than for single-level distributed computing due to MLS information flow restrictions.

When a MLS cryptographic infrastructure is designed, it should

- accommodate a variety of applications and cryptographic infrastructures in different classification domains,

- minimize encryption and other overhead, and
- accommodate multiple cryptographic mechanisms and Internet standards.

Figure 7 illustrates our proposed solution. This approach involves low-side and high-side cryptographic proxies. A low-side proxy that acts on behalf of a high-side sender or receiver understands the low-side cryptographic infrastructure. On the other hand, a high-side proxy that acts on behalf of a low-side sender or receiver understands the high-side cryptographic infrastructure.

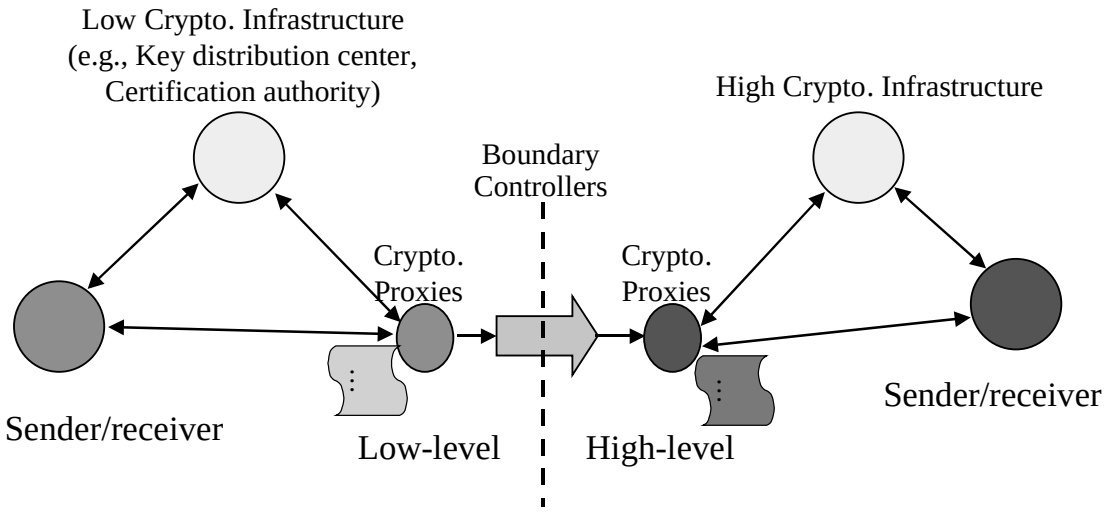

Figure 7: MLS Cryptographic services across classification domains

Consider a scenario where a low sender transmits messages to a high receiver. The high receiver needs to know if the messages come from a legitimate low source. Let's also assume that boundary controllers can use their own cryptographic algorithms that may be different from the low-side and high-side cryptographic algorithms. Let's also assume that the low-side proxy knows the set of low senders who can send messages to known high receivers. When a low sender sends a message with his own signature, that message is delivered to a low proxy. The low proxy validates the sender, through a low-level cryptographic infrastructure, that the message actually originated from the legitimate sender and is destined to a legitimate high receiver. The low proxy then relays the message to a high proxy through a boundary controller. If the high classification domain knows the low sender, the high proxy may relay the message with the low sender's signature. If the high classification domain does not know the low sender, the high proxy may relay the message with its own signature. Hence, in this case, the authentication of a low message to the high receiver is based on the trust between a low sender and a low proxy, the trust between a low proxy and a high proxy, and the trust between a high proxy and a high receiver.

As illustrated in the above scenario, cryptographic proxies can behave as if they were the endpoints (sender/receiver). The proxies perform two major roles:

- translation of the cryptographic protocol of one classification domain to the cryptographic protocol of another classification domain, and

- translation or replacement of principals so that the principal is known to the proper classification domain.

If the original message is encrypted then the problem becomes more complex. We can consider two possibilities:

- The cryptographic infrastructure of the sender's domain is replicated to the receiver's domain. In this case, encrypted messages can be passed all the way to a receiver. The receiver can decrypt the message using the replicated infrastructure.
- If the receiver's domain knows nothing about the cryptographic infrastructure of the sender's domain, then the cryptographic proxies at the sender side may have to decrypt the message. When the message reaches a receiver-side proxy, it may re-encrypt the message using the cryptographic infrastructure of the receiver's domain. Note that the message from a sender-side proxy to a receiver-side proxy may be encrypted depending upon the boundary controller and it's configuration.

3.1.4. Putting It All Together

In this subsection, we have introduced many servers, coordinators, and proxies. One may wonder how we can manage all these proxies. However, it is not difficult to see that some proxies can be combined to carry out multiple functions. Consider a scenario where a high client sends a request to a low server and expects a reply. In this case, one may want to combine a high proxy server with a high request/reply coordinator and a high cryptographic proxy as one server that deals with the boundary controllers. If each proxy that participates in the merger requires cryptographic authentication in the system design, then the duplicate function can be streamlined. *It is up to the security system designer to mix and match many different techniques and functions for their needs.* The system designers have to consider all requirements (e.g., functional, distribution, MLS, tractability) and come up with a reasonable solution that is clean and simple.

3.2. Anatomy of Classification Boundary Controllers

We have introduced the need for three MLS services for the MLS infrastructure in section 2.1. In section 3.1, we proposed solutions for the MLS infrastructure using the MSL distributed system architecture (section 2.2.1). In this section, we analyze the core functions of classification boundary controllers (CBCs) that are the key components of the proposed MLS infrastructure. Different MLS distributed architectures may have different ways to implement CBCs. However, the core functions of CBCs do not change. It is important to analyze the core functions of CBCs because it provides the basis for satisfying the MLS design goals that we described in section 1.

CBCs are, in general, high-assurance devices that have to follow rigorous development processes and go through extensive and long evaluation, certification, and accreditation processes. Hence, it is not practical to build each CBC for each specific application. Instead what we want is a high-assurance multipurpose device that can be reused for many CBCs, no matter what the application or the data.

To find if it is possible to build a high-assurance multipurpose device, A few basic design questions need to be answered.

- What basic functions does a CBC perform?
- Where does each function belong? Is the function specific to each organization or is it common to all organizations that have needs to release or receive information.
- How can we organize CBCs so that they are flexible enough to add and change functionality without affecting their trustworthiness? In other words, CBCs may consist of many building blocks that perform different functions. Do all building blocks have to be trusted in the same way? Is there any room for balanced assurance where the different building blocks can be trusted in a different way and to different degrees?

There are roughly three functional units through which information may have to pass when it goes across classification boundaries. They are a *release-policy server*, *flow controller*, and *receive-policy server*. An information releaser (sender) may have a specific policy to release information. An information receiver may have another policy to receive information. The policies may differ based on the relationship between different information senders and receivers. The flow controller makes sure that information flows only in the intended direction. Let us concentrate on each functional unit one section at a time and analyze information flow across a classification boundary in terms of functionality and policy that has to be supported.

3.2.1. Information Release

Information may pass through several functional processes before it is actually released. One process may enforce the organizational and/or application-specific release policies. A release-policy server determines if the information to be released complies with the application and organizational release policies. If there is a need to sanitize information, that process has to be performed before the information reaches a release-policy server. The flow controller enforces information flow direction (i.e., no bad message or code flows from the other side). The flow controller also makes sure that all messages it receives have been authorized by the release-policy server (i.e., all information that needs be released has to go though a valid information release-policy checker). Figure 8 shows the three functions.

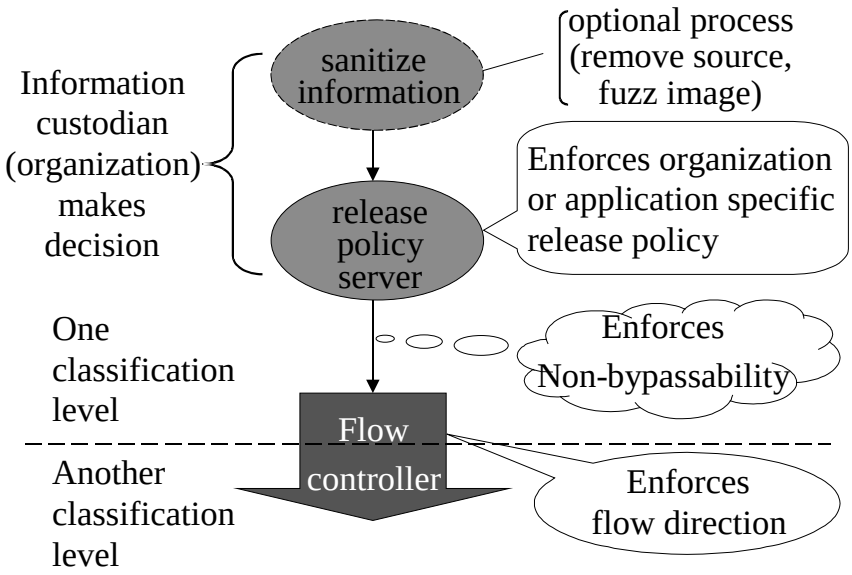

Figure 8: Anatomy of information release

In some special situations, the releaser may consider the domain to which the information is destined to be more highly protected than his domain. For a domain to be more highly protected requires that it is only accessible by more trustworthy personnel and computing resources than the releaser's domain. In this situation, the releaser's information may not need to be sanitized and the release-policy server may only have to provide an acknowledgement to the flow controller for the information's release.

Note that the answer to "which classification domain is higher (better protected) than the other?" could be very subjective. If domain A considers domain B to be accessible by suspicious personnel or computing resources, then domain A may consider itself as a higher domain than domain B. However, domain B may believe that the opposite is true (i.e., mutual suspicion).

3.2.2. Information Receive

Information receive policy depends on the trust relationship between the information sender and receiver. The receive-policy server may enforce integrity, labeling and other policies that the domain that receives information wants to enforce. For example, if the receiving organizations have specific policies such as checking for viruses or adding labels to information received from another domain, then they can implement those policies in their receive-policy servers. The flow controller makes sure only authorized information flows to the receiver's classification domain and that neither information nor bad software flows from the receiver's classification domain to the sender's classification domain. Figure 9 shows this concept.

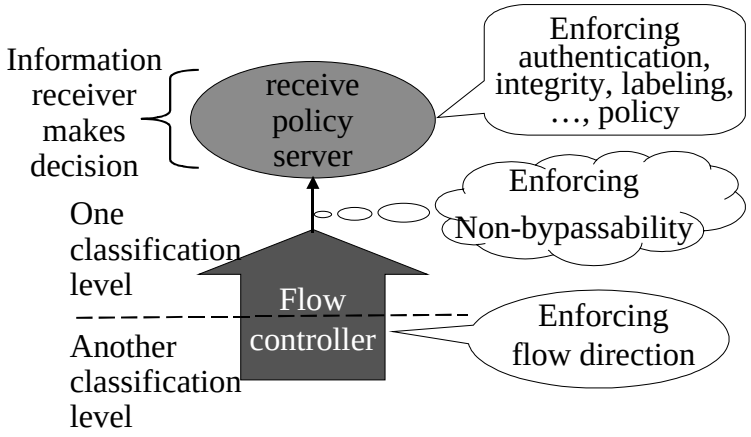

Figure 9: Anatomy of information receive

Note that if the receivers of information consider their domain a more highly classified domain than the senders' domain, then they may want to deploy a flow controller that can prevent covert information leakage such as the NRL Pump [5]. We would like to emphasize again that the strength and mechanism of policy servers and flow controllers depends on the trust relationship between releasers and receivers.

3.2.3. Implementation Issues

There are many ways to realize the functions and components in figure 8 and 9. One obvious way to implement these components is to host everything on a trusted machine or device. One advantage of this approach may be its footprint. However, we think this approach violates the MLS design principles proposed earlier in this paper. This approach may force every organization that has a need to release or receive information to maintain its own trusted machines that are usually very expensive in terms of hardware, software, and maintenance. Otherwise, this approach may create an organization whose sole purpose is to manage classification boundary controllers, which are unwanted MLS devices. This approach violates the autonomy principle and creates bureaucracy. It also forces the same level of assurance in the policy servers and the flow controller.

We propose another way to organize components in figure 8 and 9. Since each organization may have different release and receive policies, release-policy and receive-policy servers have to be updated and maintained by each organization that needs to release and receive information. The flow controller protects information in a classification domain rather than a specific organization. In addition, the policy that the flow controller enforces is a simple, invariant policy. Hence, the flow controller can be shared by many organizations in the same classification domain and does not require a bureaucracy for support. To enforce the non-bypassability property among the flow controller and policy servers, a cryptographic algorithm could be used to provide authentication and non-repudiation services. Figure 10 shows an example configuration where release-policy and receive-policy servers are managed by each organization while the flow controller is shared by many organizations.

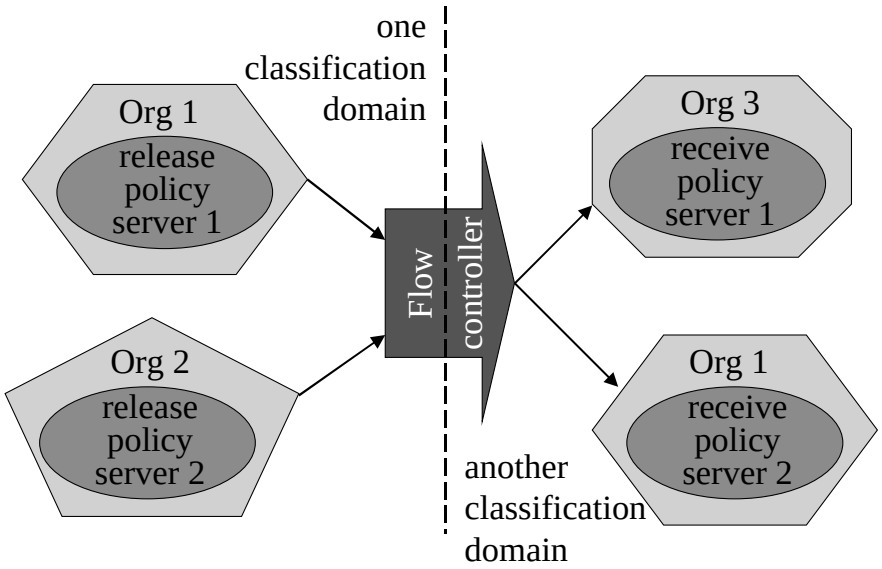

Figure 10: A configuration of information release and receive

The flow controller has to be either a trusted device or trusted software on a MLS platform because it is located at a classification boundary. The release-policy and receive-policy servers should be trusted software, where trusted means that the software will do what it is supposed to do. The question is "do we have to run a policy server on a MLS platform?" If an organization does not have to run the policy server on a MLS platform, it may save hardware and maintenance costs. We believe the policy server can be run on a single-level platform with modest trust such as C2 in the TCSEC [1] sense. The reasons are as follows:

- Since policy servers are not actually located at classification boundaries, MLS platforms do not add any additional value over single-level platforms.
- If a bad process tries to smuggle information without approval from a policy server, it must prove that the information passes the test of a policy server to a flow controller. To do this the bad process must circumvent the non-bypassability channel between a policy server and the flow controller, since a flow controller will only release information from that channel.

Therefore, what needs to be protected is access to the flow controller's communication channel. In the example above, we proposed that cryptographic algorithms be used to establish this channel. In this case, what needs protection is the cryptographic key for the user who operates the policy server. Since protecting a key on a computer is not a multilevel problem, we believe a well-engineered single-level system should do the work as effectively as a MLS system.

From the description of information release and receive, it is clear that we need a high-assurance building block, a flow controller, that is flexible enough to incorporate various mechanisms for providing a trusted communications channel without affecting the assurance argument of the device.

3.3. A Logical Design of Flow Control Devices

In section 3.2, we analyzed the classification boundary controllers that are the core components in the proposed MLS infrastructure. The main principle behind the analysis was the separation of functions and trust for tractability reasons. In that process, we identify the need for a multipurpose high-assurance flow controller that is independent of a specific application to avoid repeated evaluation and certification.

In this section, we investigate the requirements of such devices for the MSL and hybrid architectures as described in section 2.2. We can summarize the requirements of high-assurance flow controllers as follows.

- A flow controller should levy as little overhead as possible in terms of performance.
- They need to be network devices that can support many different network protocols (e.g., TCP/IP on Ethernet, Token ring, ATM) and simultaneous connections.
- They may need to incorporate cryptographic-based algorithms to support authentication and non-repudiation of policy servers, and the integrity and privacy of messages.
- They should have their own flexible protocol, which may be independent of network or application-level protocols, to avoid frequent changes of the device but, at the same time, support various network protocols.
- A flow controller should be structured so that it does not require new evaluation and certification every time a new network protocol, a new cryptographic algorithm, or even the direction that they are used is changed (i.e., upward or downward).

Figure 11 shows a configuration of the flow controller that can meet the above requirements. One of the most important guiding principles for the following configuration is “separation of function and trust” (which is the same principle that was applied to the analysis of boundary controllers), so the different components can be trusted in a different way and to different degrees. The MLS component should be small and generic to avoid repeated evaluation and the single-level components should be versatile, so they can adapt to many different environments.

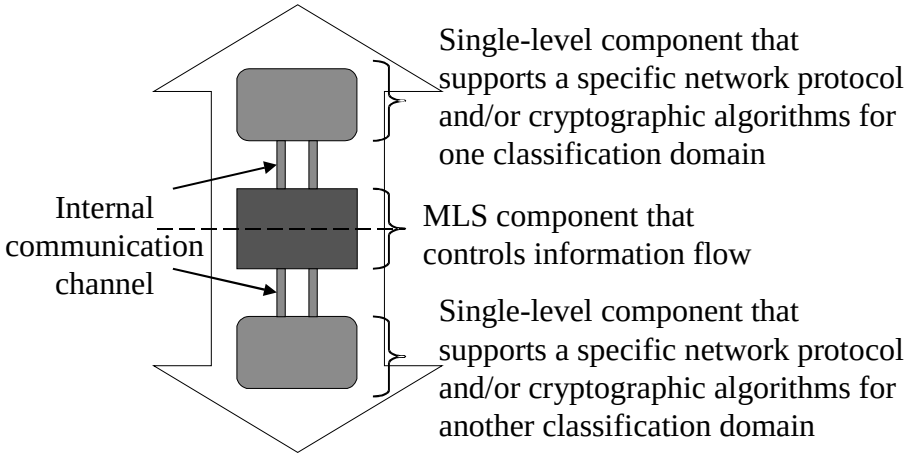

Figure 11: A multipurpose flow controller

In figure 11, the MLS middle component enforces the direction of information flow between classification domains. Two other components that are located on either side of the trusted component are single-level components that communicate to the trusted component through internal communication channels. The main functions of those two components are supporting low-level network protocols and possibly some cryptographic algorithms for the authentication of policy servers and the integrity and privacy of messages. Note that the single-level components in figure 11 can be modified without affecting the trustworthiness of the device (from the MLS point of view). Also note that there are trusted paths between the MLS component and the single-level components in the device.

4. Conclusion

For widespread use of MLS distributed computing, a MLS infrastructure that provides equivalent services and programming paradigm to the single-level distributed computing infrastructure is needed. In this paper, we examined several services for the MLS distributed infrastructure. They were MLS server activation, MLS coordination, and MLS cryptographic services. We then examined classification boundary controllers that are core pieces of the MLS infrastructure. Traditional classification boundary controllers contain many functions. In this paper, we propose to distribute organization-specific functions to policy servers where the organization can update and maintain them. This approach

1. promotes autonomy and reuse,
2. is more flexible and tractable than the traditional classification boundary controller approach, and
3. saves time and money by not forcing the same level of assurance for all components.

Finally, we examined the logical structure for a multipurpose flow controller, which may be a building block of classification boundary controllers. One of the most important aspects of such devices is separation of trust and functions. The MLS component should be as small and generic as possible to avoid repeated evaluations and the single-level components should be able to adapt to as many environments as possible.

The main reasons for the failure of MLS computing (based on the number of MLS systems that are developed and deployed) include:

- Failure to separate trust from functionality. This causes the complaint that “secure systems are expensive to build, certify, and to maintain”.
- Failure to recognize the importance of usability and to keep pace with current technology and computing paradigm. This causes the complaint that “secure systems are hard to use and unfit for performing critical tasks”.
- Failure to provide MLS infrastructure. This leads to the complaint that “secure systems are hard to build” and “security takes all the allotted time and effort that should be used for system functionality”.

We believe this paper is a step in the right direction for MLS computing in terms of trust, functionality, tractability, and usability.

References

1. Department of Defense, “Trusted computer system evaluation criteria,” DoD5200.28-STD, 1985.
2. Defense Advanced Research Projects Agency, Information Systems Office, “Security Architecture for the AITS Reference Architecture,” Draft document, 1998.
3. Gong, L. “Java Security Architecture (JDK1.2),” Draft document, 1998.
4. Kang, M. H., Froscher, J. N., and Moskowitz, I. S. “An Architecture for Multilevel Secure Interoperability,” Proceedings of 13th Computer Security Applications Conference, San Diego, CA, 1997.
5. Kang, M. H., Moskowitz, I. S. and Lee, D. C. “A network Pump,” IEEE Transactions on Software Engineering, vol. 22, no. 5, pp. 329 - 338, 1996.
6. Object Management Group “CORBA Security,” OMG document 97-02-20, 97-02-21, 1997.